МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №4

з курсу “Дискретна математика ”

Виконав:
ст. гр. КН-110
Петровський Олександр

Викладач:
Мельникова Н.І.

Львів – 2018

Тема:

«Основні операції над графами. Знаходження остова мінімальної ваги за алгоритмом Пріма – Краскала»

Мета роботи:

Набуття практичних вмінь та навичок з використання алгоритмів Пріма і Краскала

Теоретичні відомості:

Теорія графів дає простий, доступний і потужний інструмент побудови моделей прикладних задач, є ефективним засобом формалізації сучасних інженерних і наукових задач у різних областях знань.

Графом G називається пара множин (V, E), де V − множина вершин, перенумерованих числами 1, 2, ..., $n = \upsilon$; $V = \{\upsilon\}$, E − множина упорядкованих або неупорядкованих пар $e = (v', v'')$, $v' \in V$, $v'' \in V$, називаних дугами або ребрами, $E = \{e\}$. При цьому не має примусового значення, як вершини розташовані в просторі або площині і які конфігурації мають ребра.

Неорієнтованим графом G називається граф у якого ребра не мають напрямку. Такі ребра описуються неупорядкованою парою (v', v''). *Орієнтований граф (орграф)* – це граф ребра якого мають напрямок та можуть бути описані упорядкованою парою (v', v''). Упорядковане ребро називають *дугою*. Граф є *змішаним*, якщо наряду з орієнтованими ребрами (дугами) є також і неорієнтовані. При розв’язку задач змішаний граф зводиться до орграфа.

Кратними (паралельними) називаються ребра, які зв’язують одні і ті ж вершини. Якщо ребро виходить та й входить у дну і ту саму вершину, то таке ребро називається *петлею*.

Мультиграф – граф, який має кратні ребра. *Псевдограф* – граф, який має петлі. *Простий граф* – граф, який не має кратних ребер та петель.

Будь яке ребро *e інцедентно* двом вершинам (v', v''), які воно з’єднує. У свою чергу вершини (v', v'') інцендентні до ребра e . Дві вершини (v', v'') називають *суміжними*, якщо вони належать до одного й того самого ребра e, і *несуміжні* у протилежному випадку. Два *ребра називають суміжними*, якщо вони мають спільну вершину. Відношення суміжності як для вершин, так і для ребер є симетричним відношенням. *Степенем вершини* графа G називається число инцидентних їй ребер.

Граф, який не має ребер називається *пустим графом, нуль-графом*. Вершина графа, яка не інцедентна до жодного ребра, називається *ізольованою*. Вершина графа, яка інцедентна тільки до одного ребра, називається *звисаючою*.

Частина $G' = (V', E')$ графа $G = (V, E)$ називається *підграфом* графа G, якщо $V' \subseteq V$ і E' складається з тих і тільки тих ребер $e = (v', v'')$, у яких обидві кінцеві вершини $v', v'' \in V'$. Частина $G' = (V', E')$ називається *суграфом* або *остовим підграфом* графа G, якщо виконано умови: $V' = V$, $E' \subseteq E$.

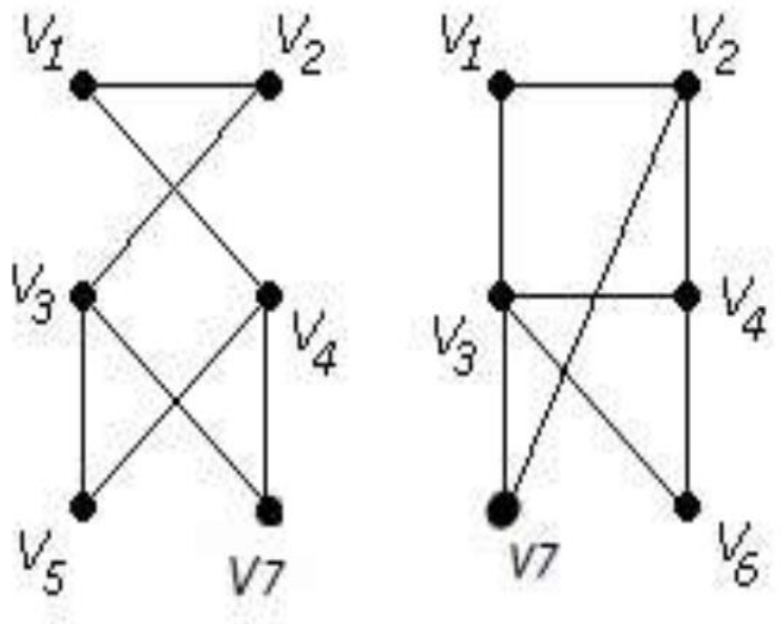
Таблицею (матрицею) суміжності [ ] ij $R = r$ графа $G = (V, E)$ називається квадратна матриця порядку n (n – число вершин графа), елементи якої ri,j (i=1, 2, …n; j=1,2, …n) визначаються наступним чином:

$$r_{ij} = \begin{cases} 1, \text{ якщо існує дуга з } \upsilon \text{ і в } \upsilon \text{ j;} \\ 0, \text{ в іншому випадку.} \end{cases}$$

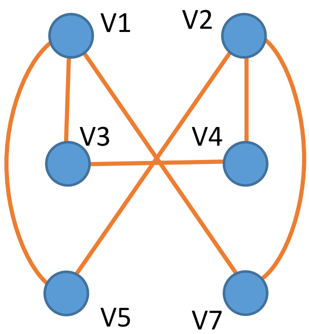
Матриця суміжності повністю визначає структуру графа.
Ексцентриситет вершини графа – відстань до максимально віддаленої від неї вершини. Для графа, для якого не визначена вага його ребер, відстань визначається у вигляді числа ребер.
Радіус графа – мінімальний ексцентриситет вершин.
Діаметр графа – максимальний ексцентриситет вершин.
Діаметром зв’язного графа називається максимально можлива довжина між двома його вершинами.

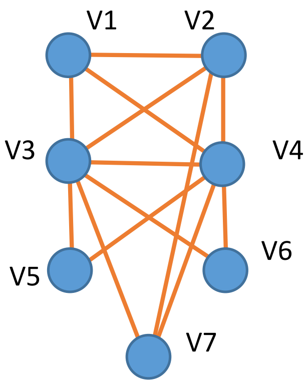
Варіант № 5

Завдання 1:

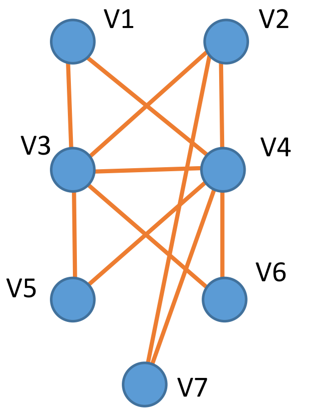


Виконати наступні операції над графами:
1) знайти доповнення до першого графу,
2) об'єднання графів,
3) кільцеву суму G1 та G2 (G1+G2),
4) розщепити вершину у другому графі,
5) виділити підграф A, що складається з 3-х вершин в G1 і знайти стягнення A в G1 (G1\ A),
6) добуток графів.

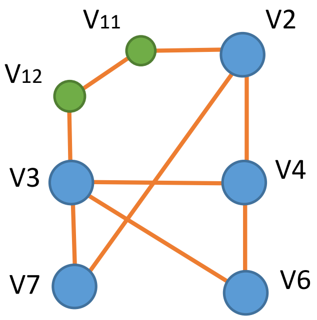
1)


2)


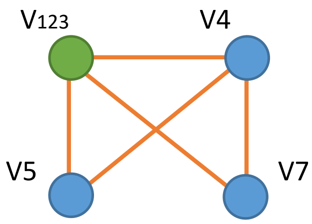
3)



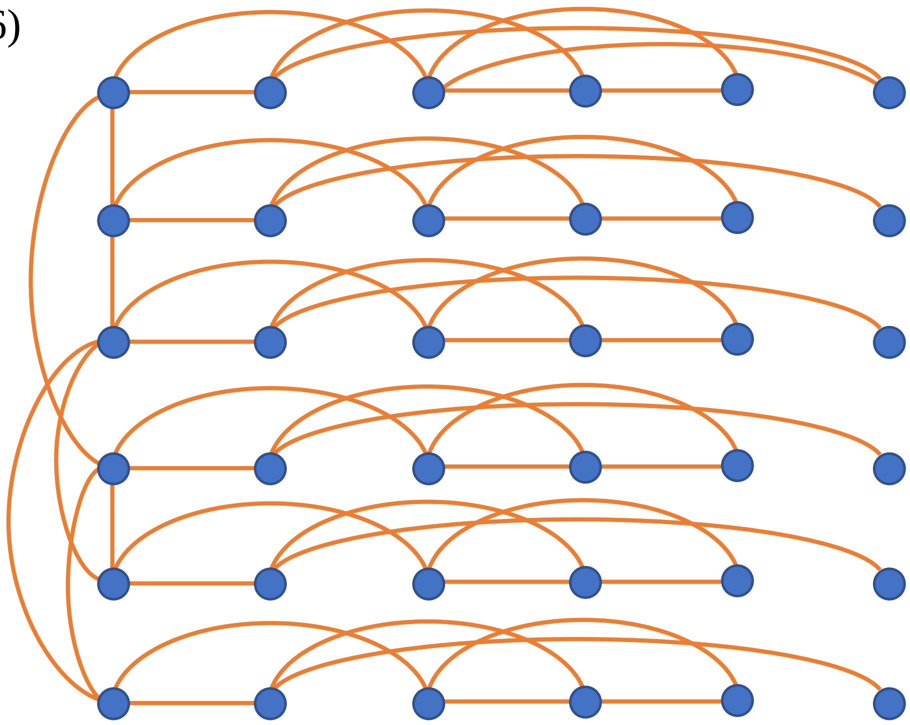
4)



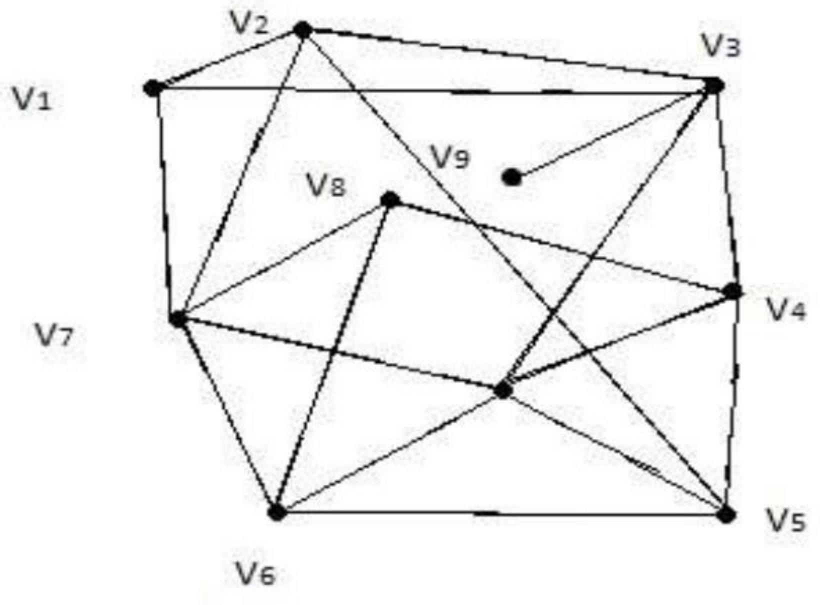
5) A = {v1, v2, v3}



6)



Завдання 2:

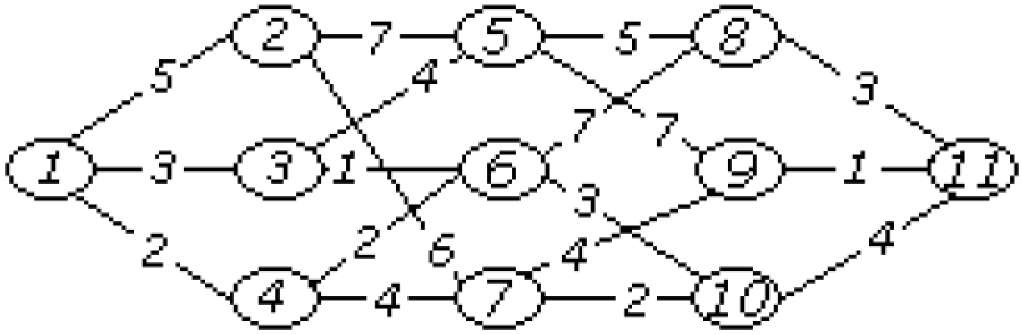


Знайти таблицю суміжності та діаметр графа.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
V1	0	1	1	0	0	0	1	0	0	0
V2	1	0	1	0	1	0	1	0	0	0
V3	1	1	0	1	0	0	0	0	1	1
V4	0	0	1	0	1	0	0	1	0	1
V5	0	1	0	1	0	1	0	0	0	1
V6	0	0	0	0	1	0	1	1	0	1
V7	1	1	0	0	0	1	0	1	0	1
V8	0	0	0	1	0	1	1	0	0	0
V9	0	0	1	0	0	0	0	0	0	0
V10	0	0	1	1	1	1	1	0	0	0

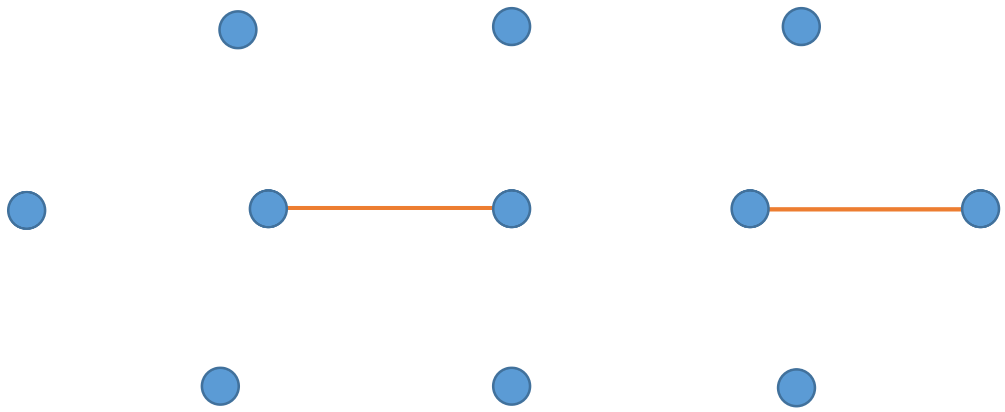
Діаметр графа дорівнює 3.

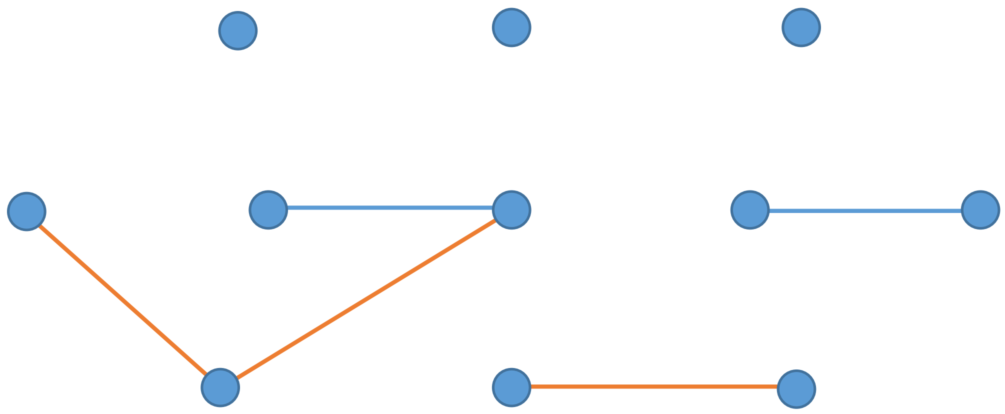
Завдання 3:

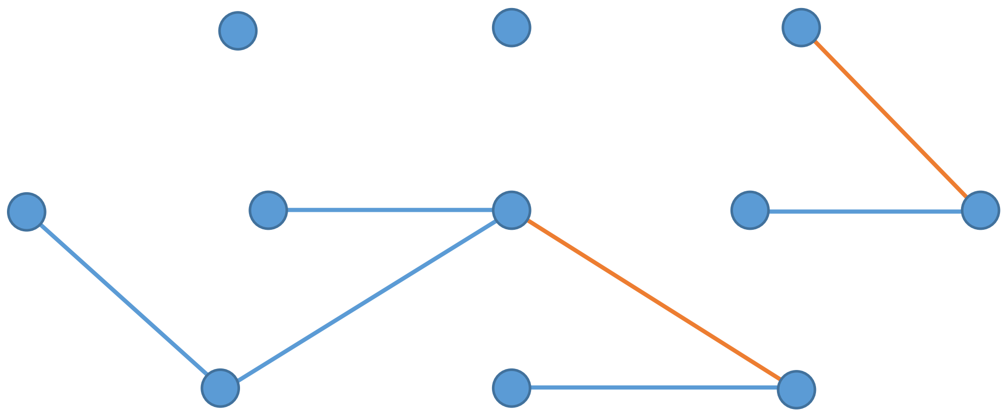


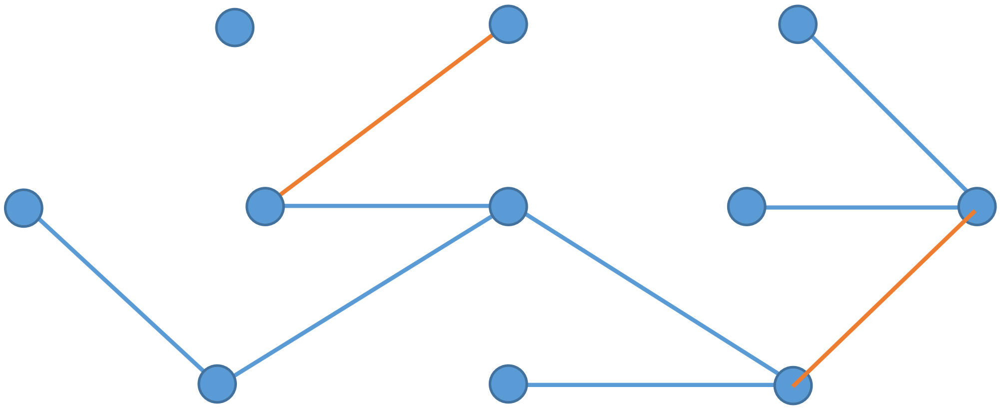
Знайти двома методами (Краскала і Прима) мінімальне остове дерево графа.

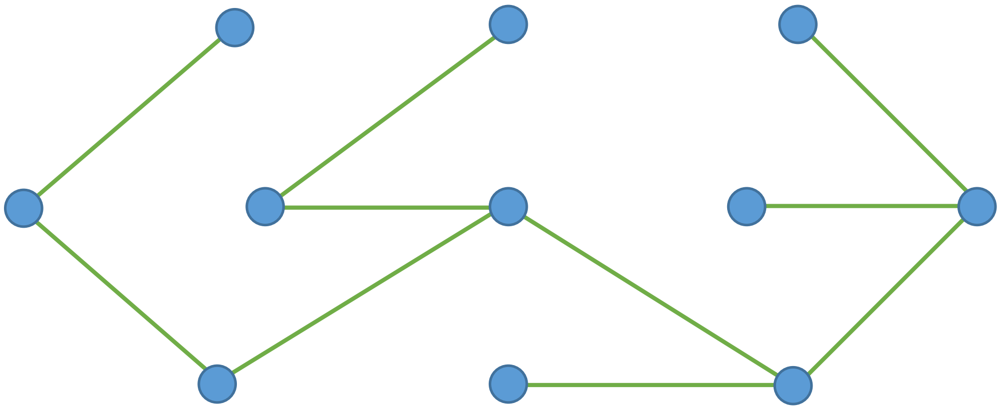
м. Краскала



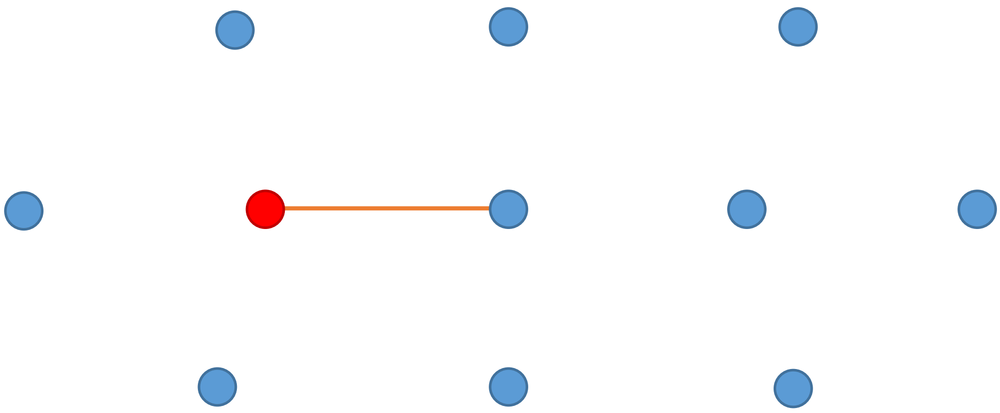


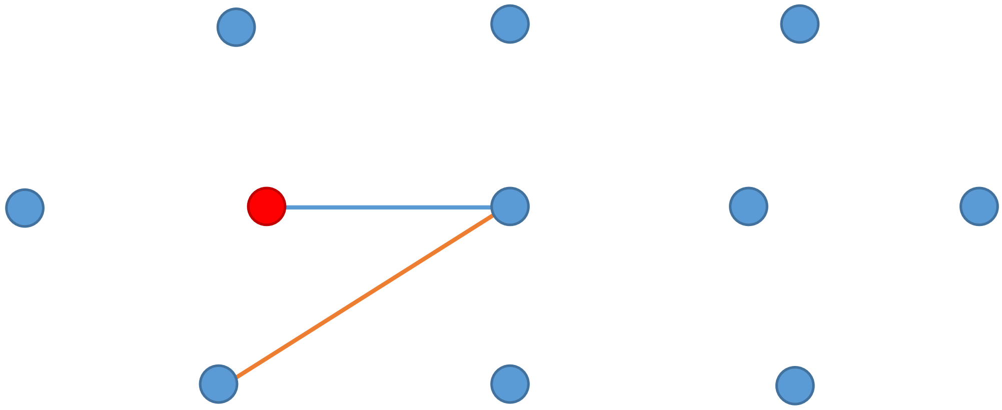


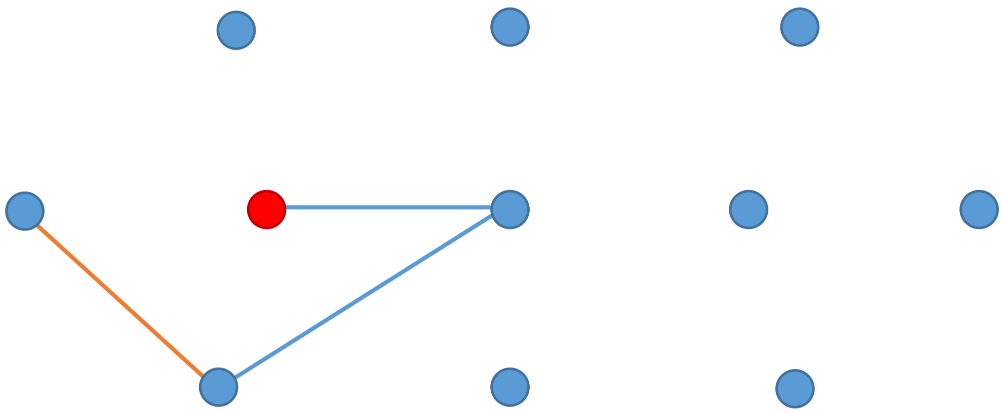


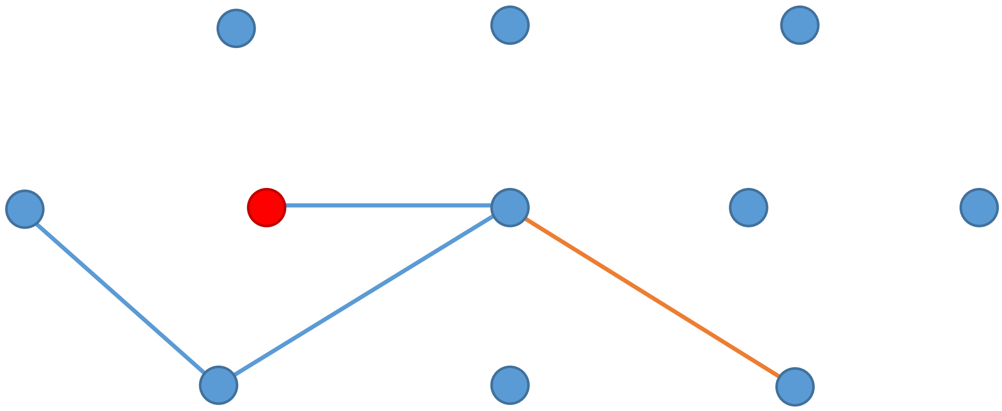


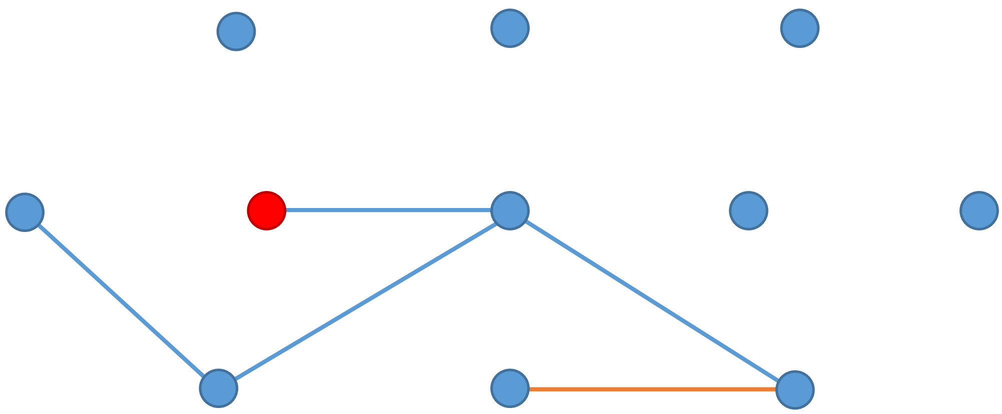
м. Прима

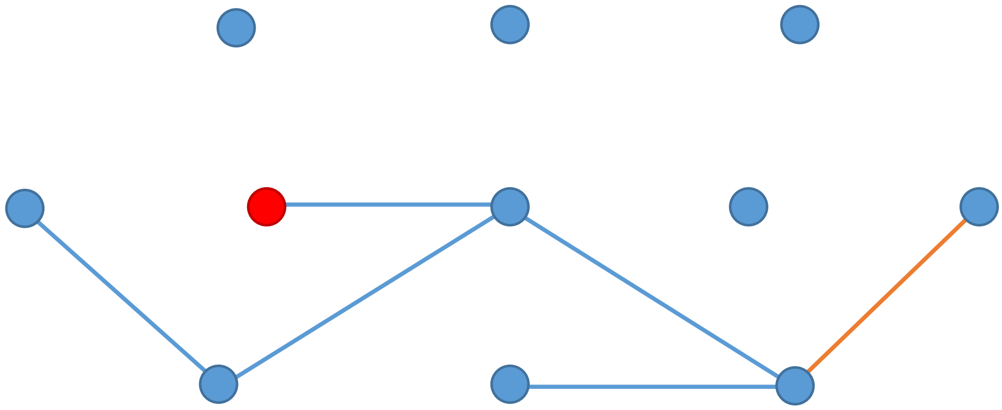


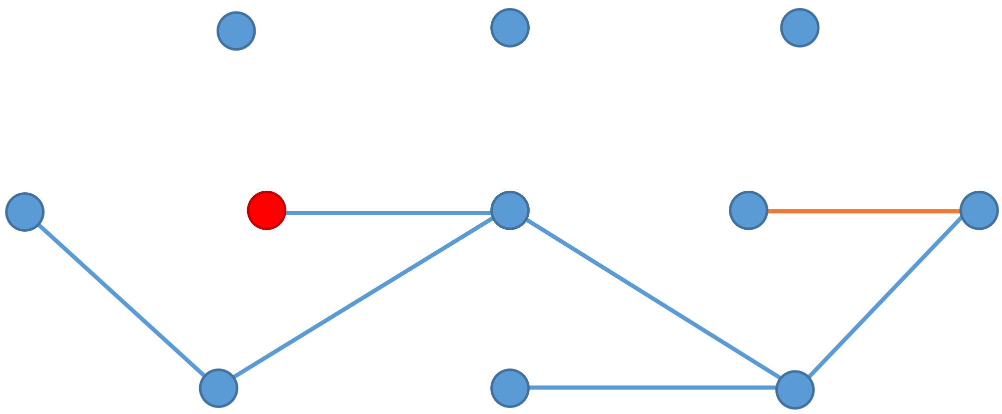


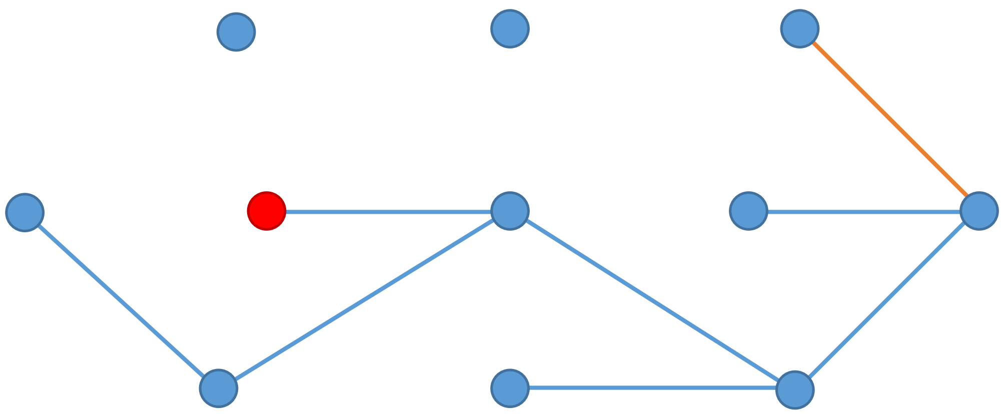


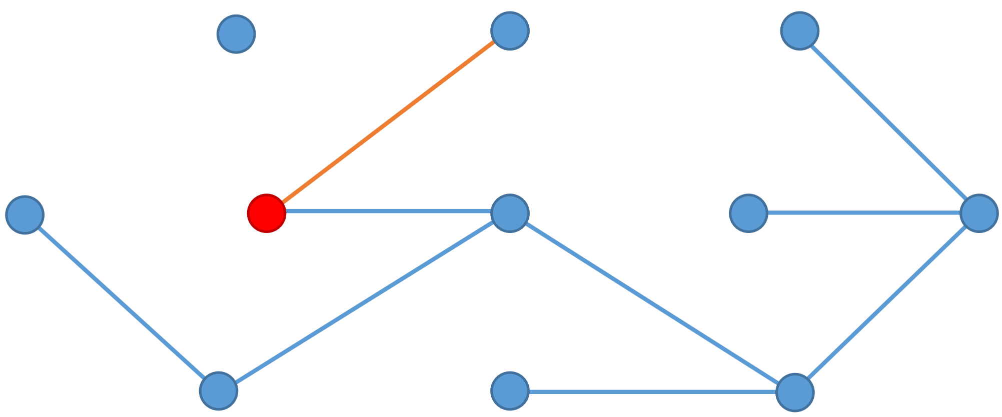


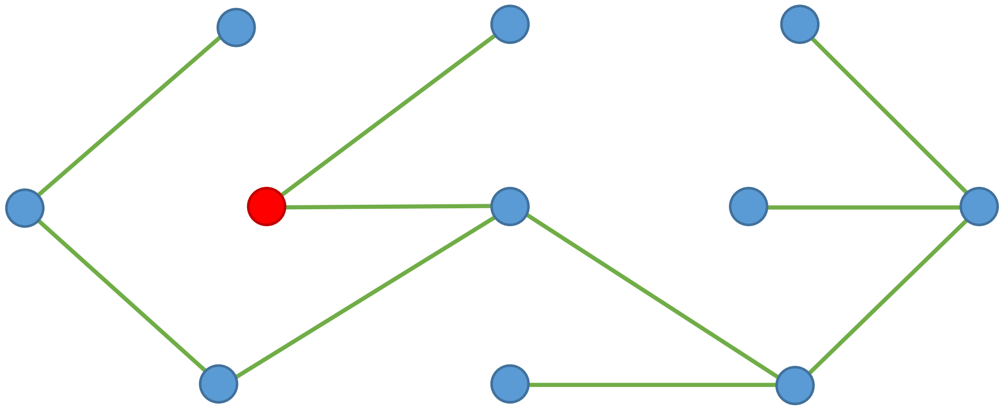






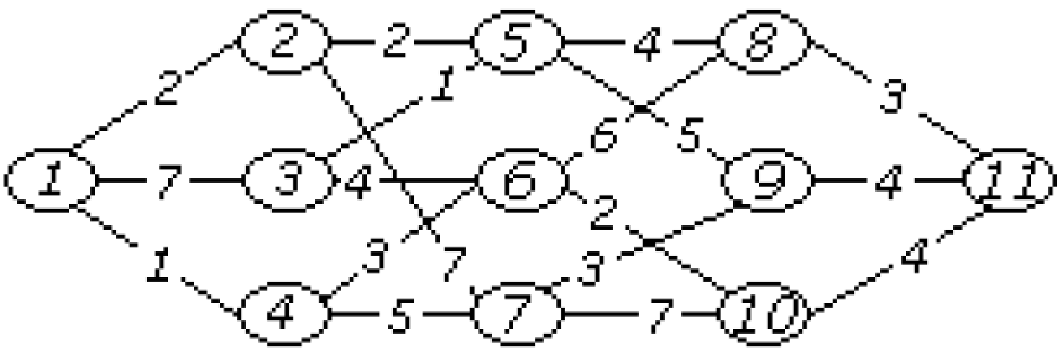






Програма:

За алгоритмом Прима знайти мінімальне остове дерево графа. Етапи розв'язання задачі виводити на екран. Протестувати розроблену програму на наступному графі:



Код:

```
#include <stdio.h>
#include <stdlib.h>

//STRUCTURES///////////////////////////////////////////////////////////
```

```
struct vert{
   short id;
   short degr;
   short used;
   struct edge *edge[10];

};

struct edge{
   short v1id;
   short v2id;
   short wght;
   short used;
   struct vert *v1pt;
   struct vert *v2pt;
};

//PROTOTYPES////////////////////////////////////////////////////////

struct vert *create(short id);

void connect(struct vert *a, struct vert *b, short weight);

void display(struct vert *start);

struct vert *get_vert(struct vert *ptr);

short search_edge(struct vert *ptr);

struct vert *search();

//VARS/////////////////////////////////////////////////////////////
```

```
short ver_am = 0;
short recuco = 0;
struct vert *list[20];
short lico = 0;

//MAIN//////////////////////////////////////////////////////////////

int main(){

//creating nodes
struct vert *v1 = create(1);
struct vert *v2 = create(2);
struct vert *v3 = create(3);
struct vert *v4 = create(4);
struct vert *v5 = create(5);
struct vert *v6 = create(6);
struct vert *v7 = create(7);
struct vert *v8 = create(8);
struct vert *v9 = create(9);
struct vert *v10 = create(10);
struct vert *v11 = create(11);

//creating edges between
connect(v1, v2, 2);
connect(v1, v3, 7);
connect(v1, v4, 1);
connect(v2, v5, 2);
connect(v2, v7, 7);
connect(v3, v5, 1);
connect(v3, v6, 4);
connect(v4, v6, 3);
connect(v4, v7, 5);
connect(v5, v8, 4);
```

```
connect(v5, v9, 5);
connect(v6, v8, 6);
connect(v6, v10, 2);
connect(v7, v9, 3);
connect(v7, v10, 7);
connect(v8, v11, 3);
connect(v9, v11, 4);
connect(v10, v11, 4);

//printing the mst
printf("||Minimum spanning tree||\n");

display(v2);

}

//FUNCTIONS////////////////////////////////////////////////////////////

//function for creating node
struct vert *create(short id){

   struct vert *vert = (struct vert*) malloc(sizeof(struct vert));

   vert->id = id;
   vert->degr = 0;
   vert->used = 0;

   ver_am++;

   return vert;

}
```

```
//function for creating edge
void connect(struct vert *a, struct vert *b, short weight){

    struct edge *edge = (struct edge*) malloc(sizeof(struct edge));

    edge->v1id = a->id;
    edge->v2id = b->id;
    edge->wght = weight;

    edge->v1pt = a;
    edge->v2pt = b;

    a->edge[a->degr] = edge;
    a->degr++;

    b->edge[b->degr] = edge;
    b->degr++;

}

//function for printing the mst
void display(struct vert *start){

if(recuco < ver_am-1){

    list[lico] = start;
    lico++;

    start->used = 1;

    recuco++;

    display(search());
```

```
}
}

//function for getting the vertex pointer
struct vert *get_vert(struct vert *ptr){

    struct vert *adr = NULL;
    short min = 100;
    short i_min;

    for (short i = 0; i < ptr->degr; i++){ if (ptr->edge[i]->wght < min && ptr-
>edge[i]->used == 0){

        if (ptr->edge[i]->v2pt->used == 0){
            adr = ptr->edge[i]->v2pt;
            min = ptr->edge[i]->wght;
            i_min = i;
        } else if(ptr->edge[i]->v1pt->used == 0){
            adr = ptr->edge[i]->v1pt;
            min = ptr->edge[i]->wght;
            i_min = i;
        }
    }
}

    ptr->edge[i_min]->used = 1;

    return adr;

}

//function for finding the edge with miniml weight
```

```
short search_edge(struct vert *ptr){

    short wgh;
    short min = 100;

    for (short i = 0; i < ptr->degr; i++) if (ptr->edge[i]->wght < min && ptr->edge[i]->used == 0) if (ptr->edge[i]->v1pt->used == 0 || ptr->edge[i]->v2pt->used == 0){
        wgh = ptr->edge[i]->wght;
        min = ptr->edge[i]->wght;
    }
    return wgh;

}

//function for printing everything that's minimal
struct vert *search(){

short min = 100;
short i_min;
struct vert *adr = NULL;

    for (short i = 0; i < lico; i++) if (search_edge(list[i]) < min){
        min = search_edge(list[i]);
        i_min = i;
    }

    adr = get_vert(list[i_min]);

    printf("(%i)-----(%i)\n", list[i_min]->id, adr->id);

    return adr;
}
```

Висновки:

Я набув практичних вмінь та навичок з використання алгоритмів Пріма і Краскала.